Milestone 2 Report - Team 03

Details about TDD process. Please document briefly in your report the experience you had using the testcases from other teams.

The tests that we got for TDD in Milestone 2 were put into the “testsM2” folder in our project.

Overall, the test cases had been provided for all functionality and in general, the test cases tested important aspects of the program. Tests for default input, errors and corner cases had been written.

However, the tests cases were not perfect. We tried fixing the tests that failed into tests that worked for us. Below are descriptions of the problems we encountered. When we could not fix the test/it did not make sense, we commented out the test or removed it.

Cat had a test that creates an infinite loop since Std.in was used to test. It would probably be better to use a different type of inputstream in order to avoid waiting for input from the user during testing.

Ls tests were not good, it was very difficult to work with them. The main problem was that the file structure was not created. Also, the file structure provided in the comments is not even correct if you look at what the tests expects as output. Furthermore, this meant that the tests could not be ran even if the files and folders were created because the paths were all wrong. There were also some tests that were just simply incorrext, it had the wrong expected output. Some test names were also wrong, for example “testRun_recursiveFlagWithMultipleArgumentSomeOfThemDoesNotExist_shouldThrowException”. In many test cases mockito is used wrongly, listFolderContent() cannot be called with args like “-R”, it is supposed to take true and false and “-R“ is not a folder.

In the Find tests, the files should not just be created randomly and removed outside of the project folder. Please do it in a specific folder within the same project because you don’t want to change the file structure for people on their own machine. There were also some minor errors here like using assertsEqual instead of assertThrows for testing exception catching (for example in testFindFolderNotExistsContent(), testRunOutputFolderNotExistsResult() and testRunOutputMixedResults()).

One of the Cat a test still had fail() in it, so that was commented out by us. Exceptions were not tested the correct way, for example in testCatFilesWithDirectory() and testCatFilesWithMissingFile(). You cannot use assertTrue, should use assertThrows. Furthermore, the mocking done is not even a mock since the real method is called every time so why use it? Also, we could not use the paths to the resources, should use File.separator to make sure files can be found on all systems.

In the date test, the date should be able to run without additional args. Also, not correct that an exception should be thrown because the format is not of the existing ones. In the unix shell it just echoes if it is not a format.

Echo also had some faulty tests. Stdout was null but the test still did stdout.toString(). Also, in our echo application we did not assume any options for echo.

For grep, exceptions need to be tested with assertThrows. For example in testCaseInsensitiveMultipleFilesWithNonExistentFile(). Furthermore, we do not print the filename before the line since grep does not do that. So, this was removed from the expected output. Empty pattern should match everything, not return blank. You should also not be able to grep a directory.

In wc, 1 line is not equal to 0 lines in a file. So, had to change some asserts in the tests we got. Here, exceptions were also tested without assertThrows.

For sed the following was noted:

- The other team uses –1 as the default value of index, our team uses 1 instead (because it matches the real situation).
- The project document does not mention that it should be able to handle multiple files, so our team did not implement that functionality.
- It seems like our two teams implement sed application based on different assumptions. The other team sets separator behind “s” directly, our team thought it should ignore those spaces, now we have implemented this function following the new standard.
- The testcases don’t use the NEWLINE of system, so we have fixed that.
- RunInsufficientDelimitersTest is confusing and seems invalid, so we deleted it.
- runMoreThanOneFileArgumentTest() should not work since sed only takes one file.

For sort, in runOptionNothingTest() and in runMultipleFilesNoOptionTest(), we think It’s better not using –nrf arguments, so we change the case into sortFromFile(f,f,f). Second is about a type error in the line 66 with ‘nfn’, we think it’s probably the ‘-nfn’ and change it. Third, we use System.lineSeparator in windows, it means ‘\r\n’ but not ‘\n’, so when we used the SortApplicationSortingHelper, we must change the \n lineSeparator into \r\n. Fourth, It’s still have some ArrayIndexOutOfBoundsException in SortApplicationSorting, so use some txt file answer instead.

In summary, the tests had to be improved a lot by our team. However, we strongly believe that our test suite is a lot better now compared to the last milestone. The new tests helped us find many bugs and corner cases that we did not think about before.

Integration testing – plan and execution

We divided our program into the following parts: Shell - parser - command – app.

Our plan is then as follows. Firstly, we try to test within the levels. For example, test different apps with each other. Then, we continue by creating test cases that test the communication between different levels of the program. These include command-app and shell-parser-command-app test interaction.

When selecting what parts to test together we tried testing things that are relevant to use together. For example, during app-app testing we test “ls” with “mkdir”.

Out integration tests are in the folder “testIntegration”.

System testing

Build artifact (runnable .jar file) in Intellij by clicking Run -> Build. Artifact has been set to build on project build.

In Unix you then run “sh systemTests/test.sh”.

Explanation of the system test:

*.template files eg. systemTests/expected/run_and_exit.template files are python string formats. systemTests/generate_expected_out.py inserts machine dependent variables (working directory, etc.) into templates, generating *.out files. (Python 3). Input tests exist in systemTests/commandLines. Input files are looped and run as input for out/artifacts/CS4218_Shell_2019_jar/CS4218-Shell-2019.jar.

Actual output is in systemTests/actual and compared with expected output.

All output and actual file names correspond to its respective test input file.

Automatic testing - Randoop

We decided to test out Randoop to automatically generate tests for our applications. In total, Randoop was used to generate about 20,000 tests. Randoop did not create any failing test cases. However, the tests that were generated are used for regression testing. Unfortunately, Intellij cannot seem to handle all testing files created by Randoop. It would also take too long to run all the tests for us when we develop. Therefore, in our submission, we only kept one of the test files from Randoop. This test file contains some of the more complex tests that we found, all trivial tests and tests we found that we had already have were removed. So, when a team member submits a new pull request, we make sure that the tests are still passed in this file. In the project this file is located at "testRandoop/RegressionTest0.java".

Other information

In order to run all tests (except system test), right click on the "test" folder and run all tests.

We have now fixed PMD properly in this milestone.

We now have two resource files since the other team used a different one for their tests. So, they are "resources" (our team's resources) and "TestResources" (which they used). However, the other team did not us their resource folder properly and a lot of input files used for tests were created outside of it.